Conway's Life with FFT

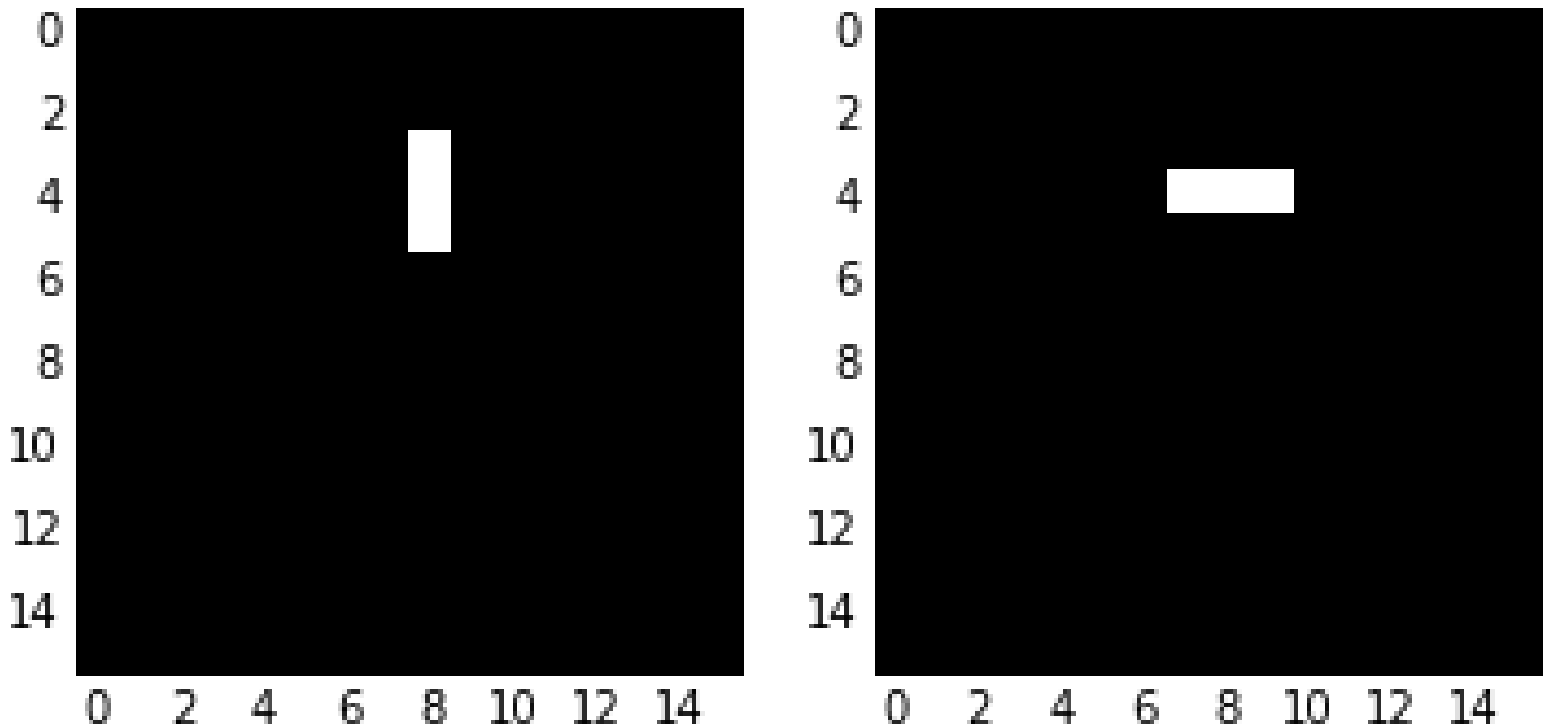
Conway's life is really a set of rules on linear filters applied to a 2D pixel grid.

We can implement these filters using FFT.

```
center = zeros((w,h))
center[0,0] = 1
surround = zeros((w,h))
surround[:3,:3] = 1
surround = roll(roll(surround,-1,0),-1,1)
image = 1.0*(rand(w,h)>0.5)
```

```
center_fft = fft2(center)
surround_fft = fft2(surround)
def life(image):
    c = abs(ifft2(center_fft*fft2(image)))
    s = abs(ifft2(surround_fft*fft2(image)))-c
    c = floor(0.5+c)
    s = floor(0.5+s)
    return AND(s>=2,AND(s<=3,OR(c,s==3)))
```

```
def blinker(size=(w,h)):
    s = zeros(size); s[:,:] = 0; s[3:6,8] = 1; return s

image = blinker()
subplot(121); imshow(image[:16,:16].copy())
image = life(image)
subplot(122); imshow(image[:16,:16].copy())
```



```
1 _=animate(iterate(life,random(0.5)),n=50)
```

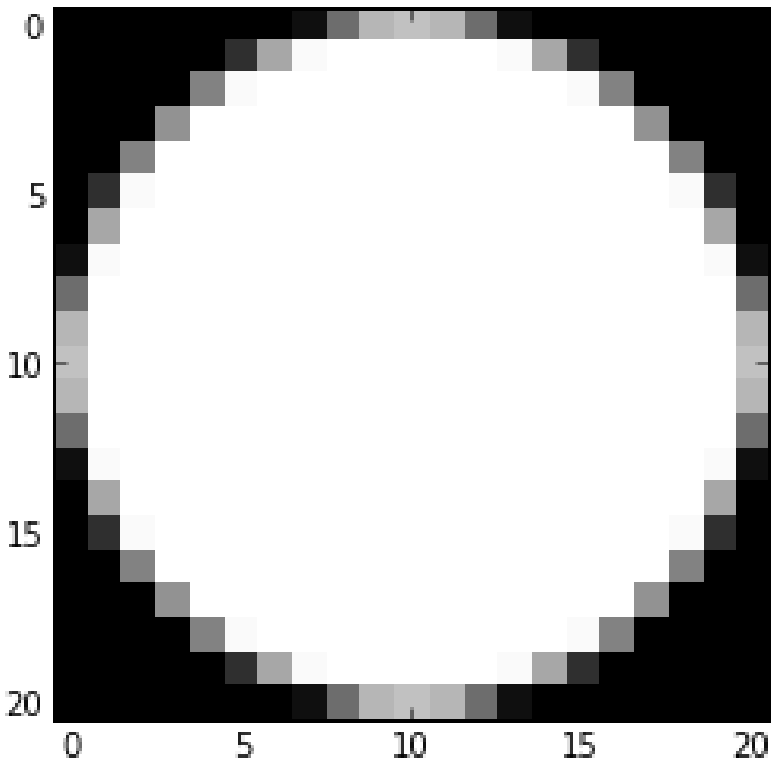
SmoothLife with FFT

While Conway's life and LargerThanLife variants use square neighborhoods and can be efficiently implemented using separable filters, circular neighborhoods (other than Gaussian) are not separable.

FFT-based implementations allow us to implement life variants with neighborhoods that aren't implementable by separable filters efficiently.

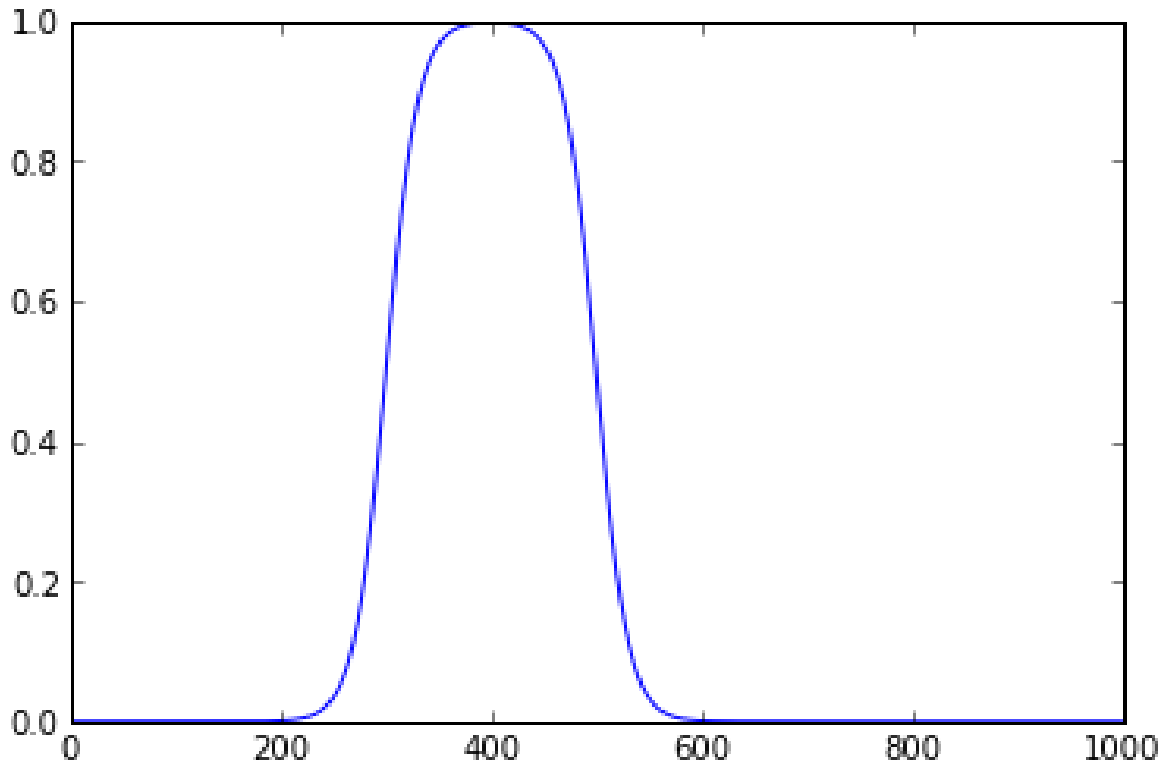
In addition, they also allow non-binary weights.

```
from scipy.ndimage import interpolation
def make_circle(r):
    s = 7
    x,y = mgrid[s*-r:s*r+1,s*-r:s*r+1]
    g = 1.0*(x**2+y**2<=(s*r)**2)
    g = filters.uniform_filter(g,s)
    return g[::s,::s]
imshow(make_circle(10))
```

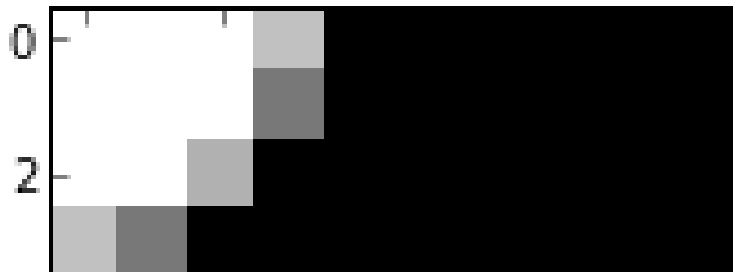


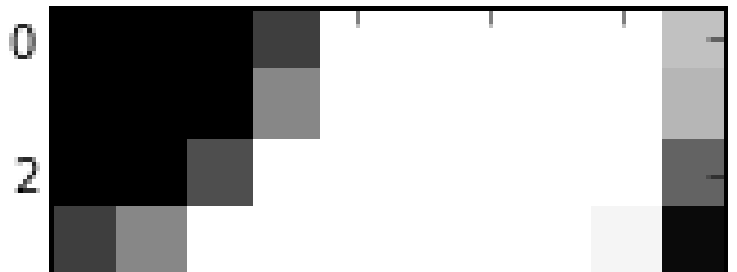
```
def normalize(s):
    return s/sum(s)

def make_fft_circle(r,size=(w,h)):
    w,h = size
    out = zeros((w,h))
    c = make_circle(r)
    cw,ch = c.shape
    out[:cw,:ch] = c
    return roll(roll(out,-(cw//2),0),-(ch//2),1)

def make_fft_annulus(r1,r2,size=(w,h)):
    outer = make_fft_circle(r2)
    inner = make_fft_circle(r1)
    return clip(outer-inner,0,1)


subplot(121); imshow(make_fft_circle(100))
subplot(122); imshow(make_fft_annulus(100,200))
```

```
1 def between(x,a,b,r):
2     return 1/((1+exp((a-x)/r))*(1+exp((x-b)/r)))
3 plot(between(linspace(0,1,1000),0.3,0.5,0.015))
```



```
class Fail(Exception): pass
ri = 3
ro = 9
scenter = normalize(make_fft_circle(ri))
scenter_fft = fft2(scenter)
ssurround = normalize(make_fft_annulus(ri,ro))
ssurround_fft = fft2(ssurround)
a0,b0,b1,a1 = 0.28,0.35,0.4,0.55
alpha = 0.003
def slife(image):
    if amax(image)<1e-4: raise Fail()
    c = abs(ifft2(scenter_fft*fft2(image)))
    s = abs(ifft2(ssurround_fft*fft2(image)))
    #image[:,:] = OR(AND(c<0.5,AND(s>b0,s<b1)),AND(c>=0.5,AND(s>a0,
        s<a1)))
    return where(c<0.5,between(s,b0,b1,alpha),between(s,a0,a1,alpha
        ))
subplot(121); imshow(scenter[:10,:10])
subplot(122); imshow(ssurround[:10,:10])
```





Random Parameters

```
for i in range(20):
    a0,b0,b1,a1 = sorted(rand(4))
    params = array([a0,b0,b1,a1])
    try:
        result = animate(iterate(slife,random(0.5)),n=50)
    except Fail:
        continue
    if mean(result)<1e-3 or mean(result)>0.3: continue
    print mean(result),params
```

```
0.10081145826 [ 0.40636815  0.4586049   0.64642207  0.89306431]
0.0243488900764 [ 0.2644781   0.40683296  0.79383081  0.91633703]
```

Life-Like Parameters with Gliders

```
for i in range(20):
    a0 = unif(0.245,0.255)
    b0 = unif(0.280,0.290)
    b1 = unif(0.39,0.42)
    a1 = unif(0.44,0.47)
    params = array([a0,b0,b1,a1])
    try:
        result = animate(iterate(slife,random(0.5)),n=50)
    except Fail:
        continue
    if mean(result)<1e-3 or mean(result)>0.3: continue
    print mean(result),params
```

```
0.00449450150111 [ 0.24731086  0.28674115  0.41918876  0.46596997]
0.00447084281993 [ 0.24587848  0.2883219   0.40875997  0.44475573]
0.17862272475 [ 0.25023591  0.28159165  0.41948315  0.46710367]
0.0699507929075 [ 0.24543855  0.2809812   0.40231587  0.46784915]
0.0122509081013 [ 0.25090391  0.2870259   0.41002448  0.4592481 ]
0.0242362225804 [ 0.25096094  0.28544712  0.41058859  0.46782726]
0.00280063783049 [ 0.24782123  0.28918379  0.41602429  0.4631311 ]
```

SmoothLife Variants

SmoothLife

- cicular center-surround neighborhoods
- anti-aliased borders
- anti-aliased / fuzzy update rules
- integral equation + difference equation (in time)
- continuous in space, discrete in time
- different from RealLife in that center has finite size
- http://www.youtube.com/watch?v=KJe9H6qS82Isvt

Differential SmoothLife

- similar to SmoothLife
- update rules give derivative rather than absolute value
- integral equation + differential equation
- continuous in space and time
- http://www.youtube.com/watch?v=TbBnHIL-R7Q
- http://www.youtube.com/watch?v=rG9UXI8IS-o